**WEEK – 7**

**REACT**

**9. REACTJS-HOL**

**Features of ES6 (ECMAScript 2015)**

* Block-scoped variables using let and const
* Arrow functions for shorter function syntax
* Template literals using backticks and ${}
* Class-based object-oriented programming
* Class inheritance using extends and super
* Destructuring of arrays and objects
* Default parameters in functions
* Rest and spread operators (...)
* Modules using import and export
* Promises for asynchronous operations
* Enhanced object literals
* Set and Map collections for data structures

**JavaScript let**

* Declares block-scoped variables
* Does not hoist like var
* Can be reassigned

Example:

let score = 10;

score = 20; // Valid

**Difference between var and let**

|  |  |  |
| --- | --- | --- |
| **Feature** | **var** | **let** |
| Scope | Function-scoped | Block-scoped |
| Redeclaration | Allowed | Not allowed in same scope |
| Hoisting | Yes (initialized as undefined) | Yes (but not initialized) |
| Global Object | Adds to window | Does not add to window |

**JavaScript const**

* Declares block-scoped constants
* Must be initialized during declaration
* Cannot be reassigned
* Internal values of arrays/objects can be changed

Example

const name = "Virat";

const arr = [1, 2, 3];

arr.push(4); // Allowed

**ES6 Class Fundamentals**

* Use class keyword to define object blueprints
* Includes constructor() and methods

Example

class Player {

constructor(name, score) {

this.name = name;

this.score = score;

}

display() {

return `${this.name} scored ${this.score}`;

}

}

**ES6 Class Inheritance**

* Use extends to inherit from a base class
* Use super() to call the parent constructor

Example

class Cricketer extends Player {

constructor(name, score, team) {

super(name, score);

this.team = team;

}

details() {

return `${this.name} plays for ${this.team}`;

}

}

**Arrow Functions in ES6**

* Shorter syntax for function expressions
* Does not bind its own this\

Example

const greet = name => `Hello, ${name}`;

const add = (a, b) => a + b;

**Set() in ES6**

* Collection of **unique** values
* Automatically removes duplicates

Example

const numbers = new Set([1, 2, 2, 3]);

// Output: Set(3) {1, 2, 3}

**Map() in ES6**

* Collection of key-value pairs
* Keys can be of any type

Example

const scores = new Map();

scores.set("Virat", 85);

scores.set("Rohit", 90);

console.log(scores.get("Virat")); // 85

**APPROACH**

1. Create the React app using npx create-react-app cricketapp and open it in VS Code.
2. Inside the src folder, create two component files: ListofPlayers.js and IndianPlayers.js.
3. In ListofPlayers, declare an array of 11 players with names and scores, display them using map(), and filter players with scores below 70 using filter() and arrow functions.
4. In IndianPlayers, separate players into odd and even teams using array destructuring, then merge T20players and RanjiTrophyPlayers arrays using the spread operator.
5. In App.js, use a flag variable and render either ListofPlayers or IndianPlayers using a simple if-else (ternary) expression.

**CODE**

**ListofPlayers.js**

import React from 'react';

const ListofPlayers = () => {

  const players = [

    { name: 'Virat Kohli', score: 85 },

    { name: 'Rohit Sharma', score: 60 },

    { name: 'Shubman Gill', score: 72 },

    { name: 'KL Rahul', score: 40 },

    { name: 'Hardik Pandya', score: 95 },

    { name: 'Jasprit Bumrah', score: 30 },

    { name: 'Ravindra Jadeja', score: 78 },

    { name: 'Suryakumar Yadav', score: 65 },

    { name: 'Mohammed Shami', score: 50 },

    { name: 'Ravichandran Ashwin', score: 80 },

    { name: 'Yuzvendra Chahal', score: 20 }

  ];

  const filteredPlayers = players.filter(player => player.score < 70);

  return (

    <div>

      <h2>All Players</h2>

      <ul>

        {players.map((player, index) => (

          <li key={index}>{player.name} - {player.score}</li>

        ))}

      </ul>

      <h3>Players with score below 70</h3>

      <ul>

        {filteredPlayers.map((player, index) => (

          <li key={index}>{player.name} - {player.score}</li>

        ))}

      </ul>

    </div>

  );

};

export default ListofPlayers;

**IndianPlayers.js**

import React from 'react';

const IndianPlayers = () => {

  const players = ['Dhoni', 'Kohli', 'Raina', 'Jadeja', 'Ashwin', 'Rohit'];

  const oddTeam = players.filter((\_, index) => index % 2 === 0);

  const evenTeam = players.filter((\_, index) => index % 2 !== 0);

  const T20players = ['Ishan Kishan', 'Sanju Samson'];

  const RanjiTrophyPlayers = ['Ajinkya Rahane', 'Cheteshwar Pujara'];

  const allPlayers = [...T20players, ...RanjiTrophyPlayers];

  return (

    <div>

      <h2>Odd Team Players</h2>

      <ul>

        {oddTeam.map((player, index) => (

          <li key={index}>{player}</li>

        ))}

      </ul>

      <h2>Even Team Players</h2>

      <ul>

        {evenTeam.map((player, index) => (

          <li key={index}>{player}</li>

        ))}

      </ul>

      <h2>All Merged Players (T20 + Ranji Trophy)</h2>

      <ul>

        {allPlayers.map((player, index) => (

          <li key={index}>{player}</li>

        ))}

      </ul>

    </div>

  );

};

export default IndianPlayers;

**App.js**

import React from 'react';

import ListofPlayers from './ListofPlayers';

import IndianPlayers from './IndianPlayers';

function App() {

  const flag = false; // Change to false to test other component

  return (

    <div className="App">

      <h1>🏏 Welcome to Cricket App 🏏</h1>

      {flag ? <ListofPlayers /> : <IndianPlayers />}

    </div>

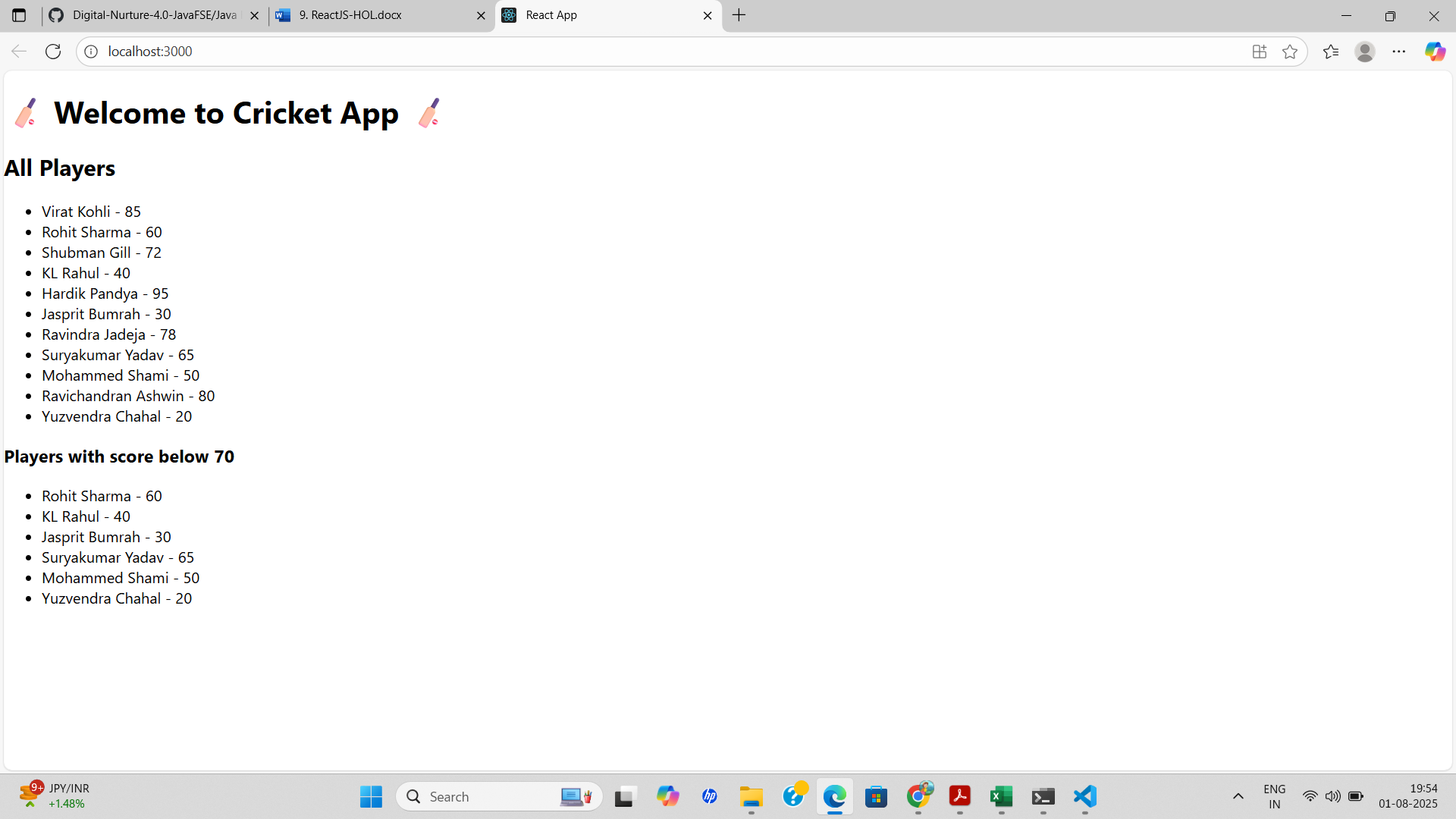
  );

}

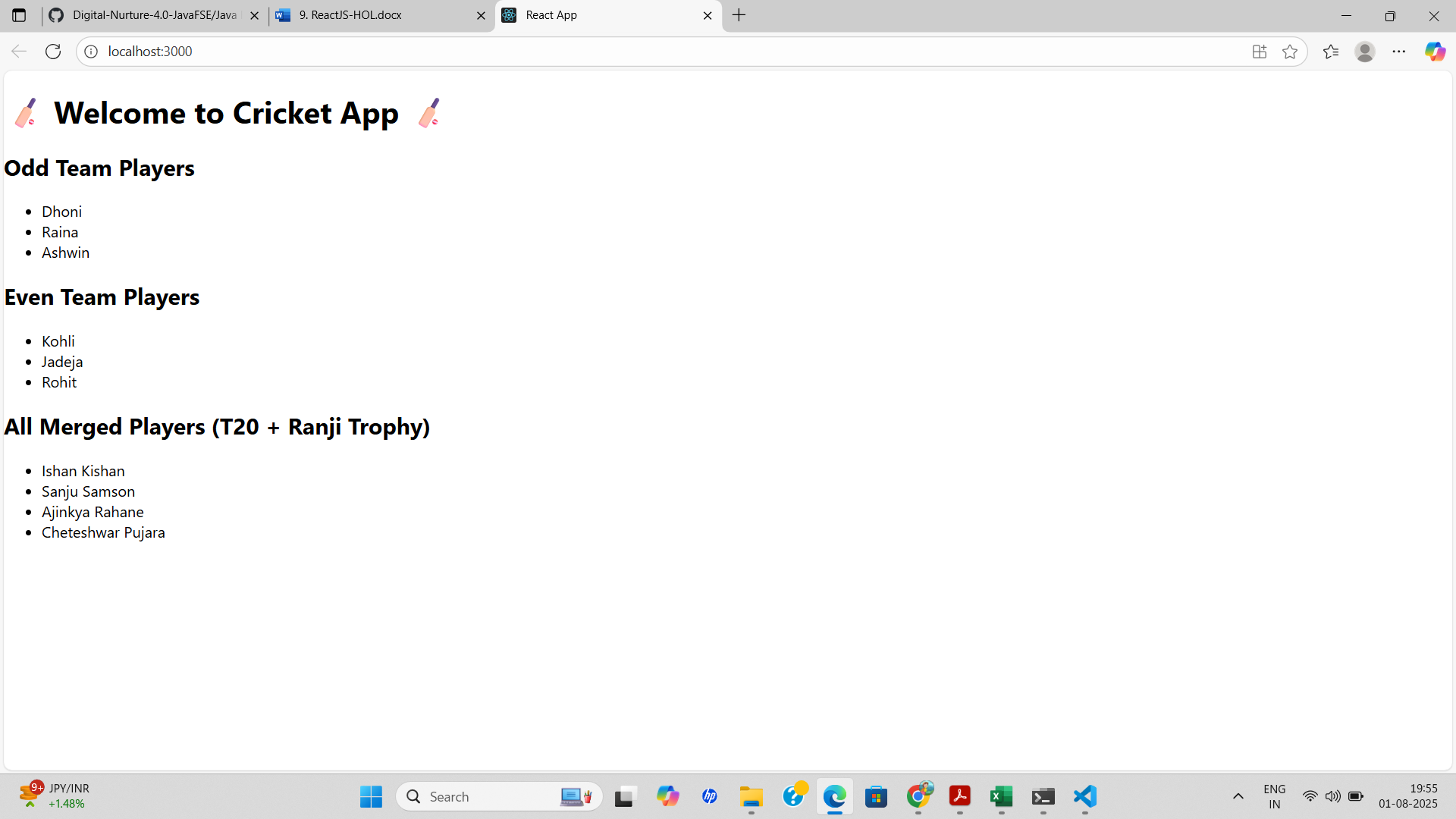
export default App;

**OUTPUT**

When flag=true, output is



When flag=false, output is



**10. REACTJS-HOL**

**Define JSX**  
JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe what the UI should look like. It allows you to write HTML-like code directly within JavaScript. JSX makes it easier to write and visualize the component structure by combining HTML and JavaScript logic in a single file.

Example:

const element = <h1>Hello, React!</h1>;

**Explain about ECMA Script**  
ECMAScript (ES) is the standard specification that JavaScript follows. It defines how the language should work and evolve. ES6 (ECMAScript 2015) introduced modern JavaScript features like let, const, arrow functions, classes, template literals, promises, destructuring, and modules, which are widely used in React applications.

**Explain React.createElement()**  
React.createElement() is the function used by React to create virtual DOM elements. JSX is syntactic sugar for this function. While JSX looks like HTML, under the hood it's converted to React.createElement() calls.

Example:

const element = React.createElement('h1', null, 'Hello World');

Equivalent JSX:

const element = <h1>Hello World</h1>;

**Explain how to create React nodes with JSX**  
React nodes can be created using JSX by writing HTML-like elements in JavaScript code. You can include elements, components, and expressions within JSX to construct the UI.

Example:

const title = <h1>Welcome to Office Rental</h1>;

const subtitle = <p>Find your perfect office space</p>;

const content = (

<div>

{title}

{subtitle}

</div>

);

**Define how to render JSX to DOM**  
To render JSX to the actual DOM, React uses the ReactDOM.render() method (in React 17 and earlier) or ReactDOM.createRoot().render() (from React 18 onward). JSX elements are passed into this function to mount the component or element on the HTML page.

React 18 Example:

import ReactDOM from 'react-dom/client';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);

**Explain how to use JavaScript expressions in JSX**  
In JSX, you can embed any valid JavaScript expression inside curly braces {}. These expressions can be variables, function calls, or calculations.

Example:

const name = "OfficeSpace";

const element = <h1>Welcome to {name}</h1>;

**Explain how to use inline CSS in JSX**  
In JSX, inline styles are written as objects using camelCase property names instead of hyphenated strings. The style object is passed to the style attribute.

Example:

const headingStyle = {

color: 'blue',

fontSize: '24px'

};

const element = <h1 style={headingStyle}>Styled Heading</h1>;

**APPROACH**

1. Run npx create-react-app officespacerentalapp and open the project in VS Code.
2. Use JSX to create a heading element and an image with attributes like src and alt.
3. Declare a single office object with name, rent, and address, and display its details using JSX.
4. Create a list (array) of office objects and loop through it using map() to display each item.
5. Apply conditional CSS classes to style rent in red if it's below 60000 and green if it's above.

**CODE**

**App.js**

import React from 'react';

import './App.css';

function App() {

  const heading = <h1>🏢 Office Space Rental App</h1>;

  const imageURL = "https://via.placeholder.com/400x200.png?text=Office+Space";

  const office1 = {

    name: "Prestige Tech Park",

    rent: 55000,

    address: "Bangalore, Karnataka"

  };

  const officeList = [

    {

      name: "WeWork Residency",

      rent: 45000,

      address: "Mumbai, Maharashtra"

    },

    {

      name: "IndiQube Alpha",

      rent: 65000,

      address: "Chennai, Tamil Nadu"

    },

    {

      name: "Smartworks HQ",

      rent: 72000,

      address: "Hyderabad, Telangana"

    },

    {

      name: "91Springboard",

      rent: 58000,

      address: "Delhi NCR"

    }

  ];

  return (

    <div className="App">

      {heading}

      <img src={imageURL} alt="Office" width="400" height="200" />

      <h2>Featured Office</h2>

      <p><strong>Name:</strong> {office1.name}</p>

      <p><strong>Rent:</strong> <span className={office1.rent > 60000 ? 'high' : 'low'}>{office1.rent}</span></p>

      <p><strong>Address:</strong> {office1.address}</p>

      <h2>Available Office Spaces</h2>

      <ul>

        {officeList.map((office, index) => (

          <li key={index}>

            <p><strong>Name:</strong> {office.name}</p>

            <p><strong>Rent:</strong> <span className={office.rent > 60000 ? 'high' : 'low'}>{office.rent}</span></p>

            <p><strong>Address:</strong> {office.address}</p>

            <hr />

          </li>

        ))}

      </ul>

    </div>

  );

}

export default App;

**App.css**

.App {

  font-family: Arial, sans-serif;

  padding: 20px;

  background-color: #f3f3f3;

}

h1,

h2 {

  color: #333;

}

img {

  margin: 20px 0;

  border-radius: 10px;

}

.low {

  color: red;

  font-weight: bold;

}

.high {

  color: green;

  font-weight: bold;

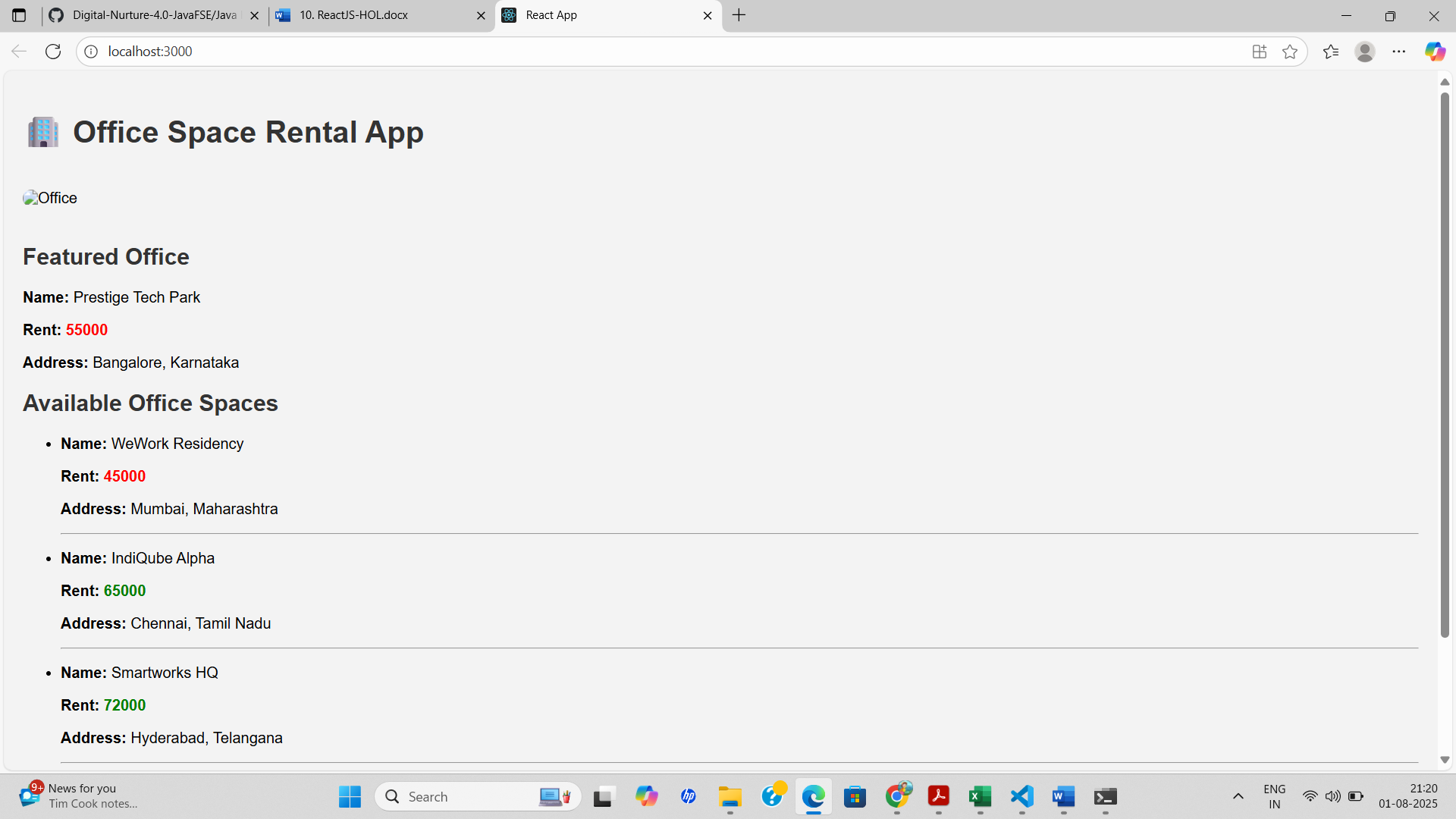
}

li {

  margin-bottom: 15px;

}

**OUTPUT**



**11. REACTJS-HOL**

**Explain React events**React events are similar to DOM events in HTML but are handled using JSX syntax. These events allow users to interact with elements such as buttons, forms, inputs, etc. React wraps native browser events into a cross-browser wrapper called a Synthetic Event, providing consistent behavior across different browsers.

Example:

<button onClick={handleClick}>Click Me</button>

**Explain about event handlers**Event handlers in React are functions that are triggered when a specific event occurs, such as a button click, form submission, or key press. These handlers are passed as props (like onClick, onChange, etc.) and are typically written as arrow functions or method references.

Example:

function handleClick() {

alert("Button was clicked");

}

<button onClick={handleClick}>Click</button>

**Define Synthetic event**  
A Synthetic Event is a cross-browser wrapper around the browser’s native event system. It combines the behavior of different browsers into one consistent interface. React uses Synthetic Events to ensure performance and compatibility. It works identically across all browsers.

Example:

function handleClick(e) {

e.preventDefault(); // using SyntheticEvent object

console.log("Synthetic Event Triggered");

}

**Identify React event naming convention**React uses camelCase for event names instead of lowercase like in HTML. Also, event handlers are passed as functions instead of strings.

|  |  |
| --- | --- |
| HTML | React |
| <button onclick=""> | <button onClick={...}> |
| <form onsubmit=""> | <form onSubmit={...}> |
| <input onchange=""> | <input onChange={...}> |

**APPROACH**

1. Run npx create-react-app eventexamplesapp and open the project in VS Code.
2. Create two components: one for handling button click events and another for currency conversion using form input.
3. Use onClick events to trigger functions for incrementing, decrementing, saying hello, passing arguments, and displaying click messages.
4. Use arrow functions to pass parameters and handle synthetic events like showing alerts when buttons are clicked.
5. Handle form submission using onSubmit to convert rupees to euros and display the result by updating state.

**CODE**

**EventExample.js**

import React, { useState } from 'react';

function EventExample() {

  const [count, setCount] = useState(0);

  // Method 1: Increment

  const handleIncrement = () => {

    setCount(prev => prev + 1);

    sayHello(); // calling another method

  };

  // Method 2: Say Hello

  const sayHello = () => {

    alert("Hello! Have a great day!");

  };

  // Decrement function

  const handleDecrement = () => {

    setCount(prev => prev - 1);

  };

  // Say Welcome with argument

  const sayWelcome = (message) => {

    alert(message);

  };

  // Synthetic Event

  const handleClick = () => {

    alert("I was clicked");

  };

  return (

    <div>

      <h2>Event Handling Example</h2>

      <h3>Counter: {count}</h3>

      <button onClick={handleIncrement}>Increment</button>

      <button onClick={handleDecrement}>Decrement</button>

      <br /><br />

      <button onClick={() => sayWelcome("Welcome to the Event Example App!")}>Say Welcome</button>

      <br /><br />

      <button onClick={handleClick}>OnPress</button>

    </div>

  );

}

export default EventExample;

**CurrencyConvertor.js**

import React, { useState } from 'react';

function CurrencyConvertor() {

  const [rupees, setRupees] = useState('');

  const [euros, setEuros] = useState('');

  const handleSubmit = (e) => {

    e.preventDefault();

    const euroValue = (parseFloat(rupees) / 90).toFixed(2);

    setEuros(euroValue);

  };

  return (

    <div>

      <h2>Currency Convertor</h2>

      <form onSubmit={handleSubmit}>

        <label>Indian Rupees: </label>

        <input

          type="number"

          value={rupees}

          onChange={(e) => setRupees(e.target.value)}

        />

        <button type="submit">Convert</button>

      </form>

      <h3>Converted Euros: €{euros}</h3>

    </div>

  );

}

export default CurrencyConvertor;

**App.js**

import React from 'react';

import './App.css';

import EventExample from './EventExample';

import CurrencyConvertor from './CurrencyConvertor';

function App() {

  return (

    <div className="App">

      <h1>Event Examples App</h1>

      <EventExample />

      <hr />

      <CurrencyConvertor />

    </div>

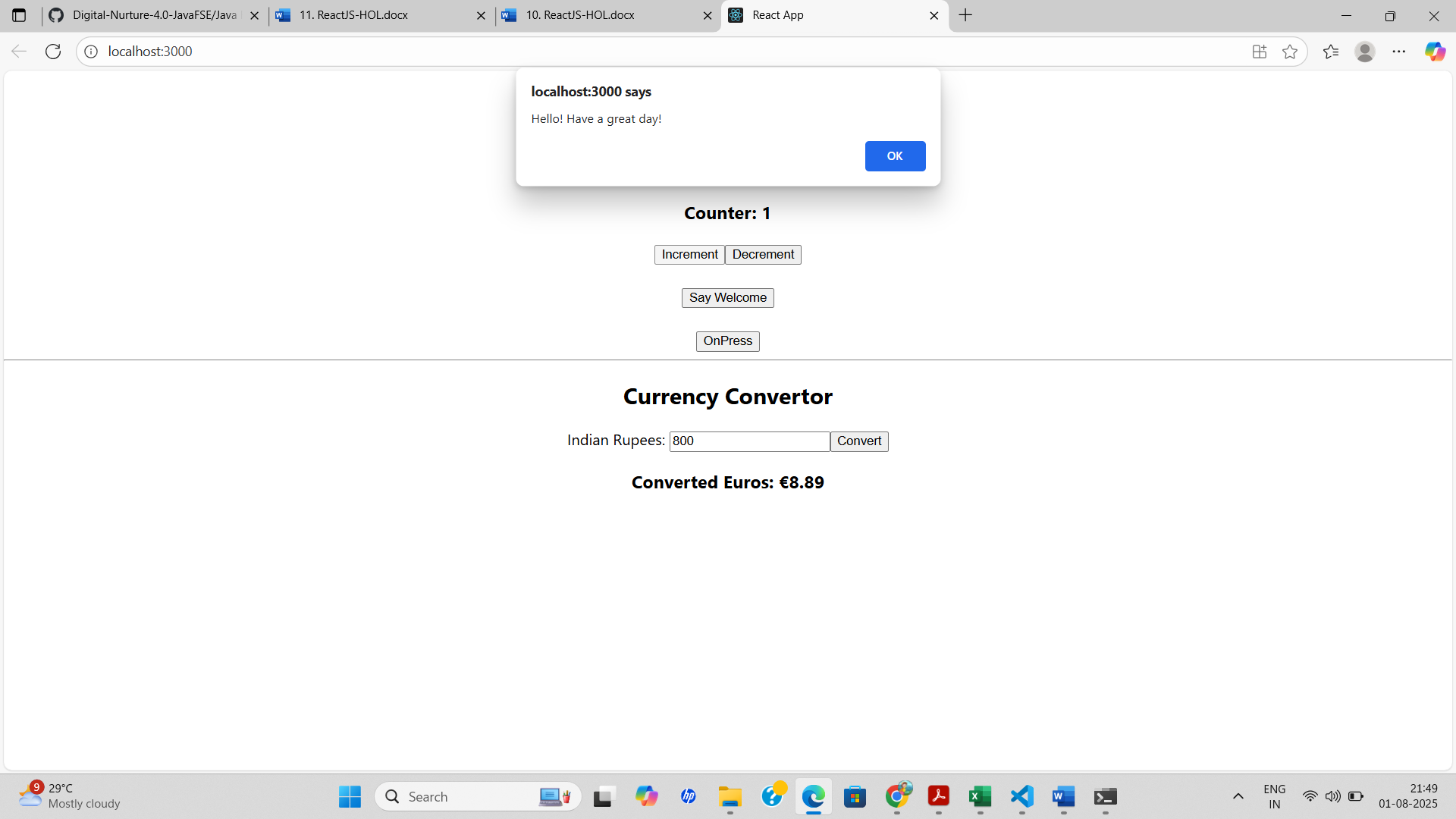
  );

}

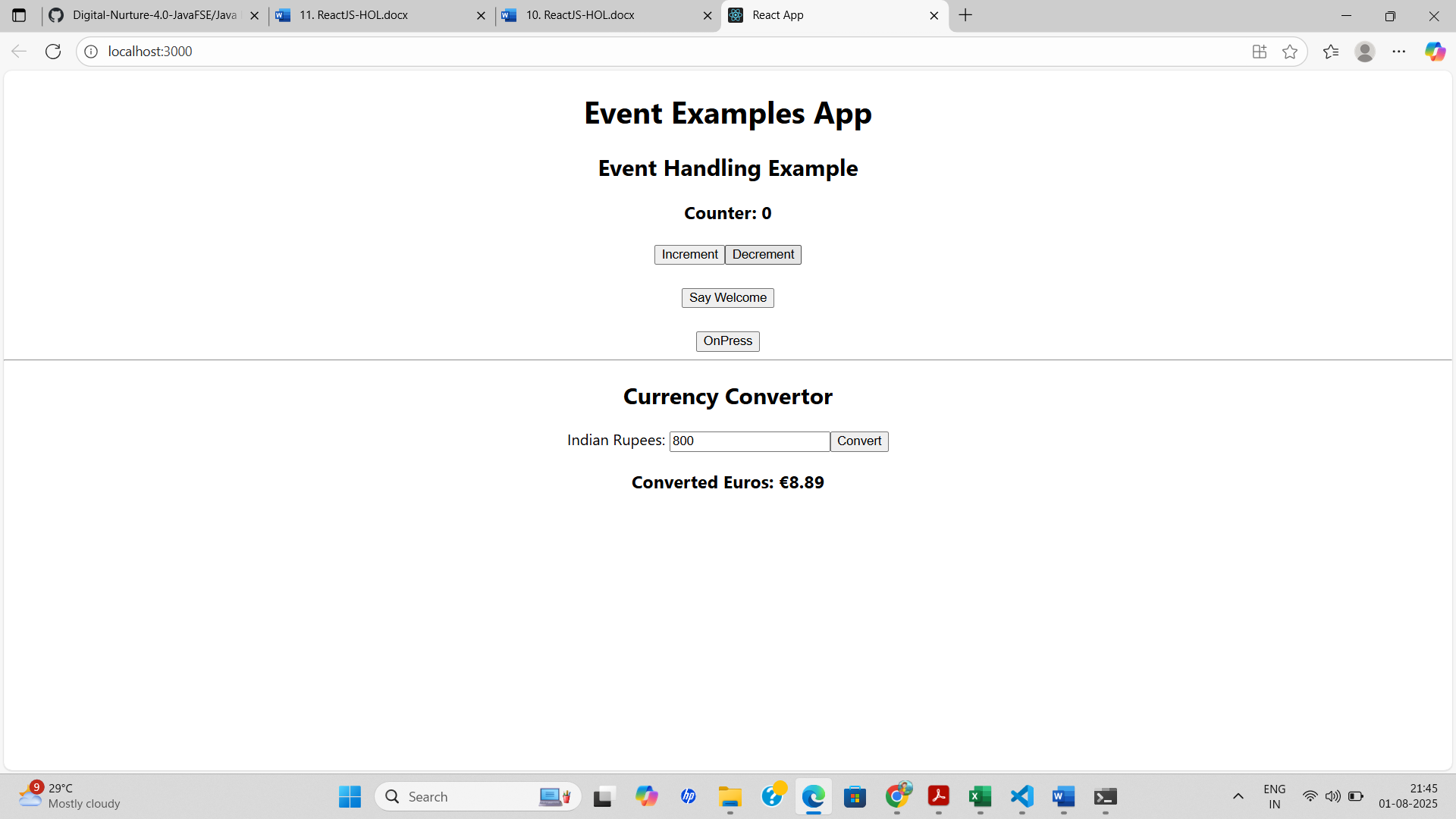
export default App;

**OUTPUT**

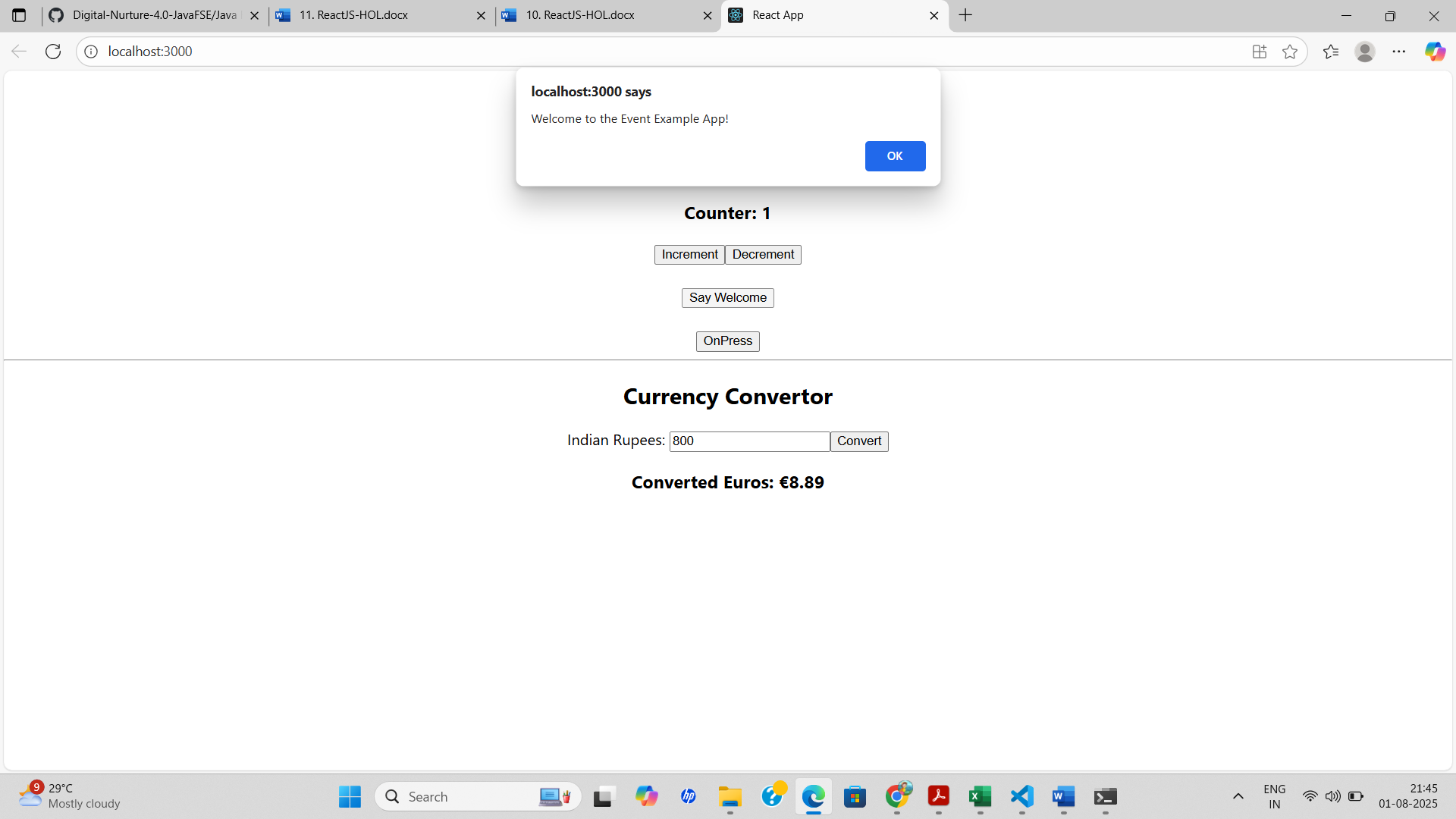
When ‘Increment’ was clicked



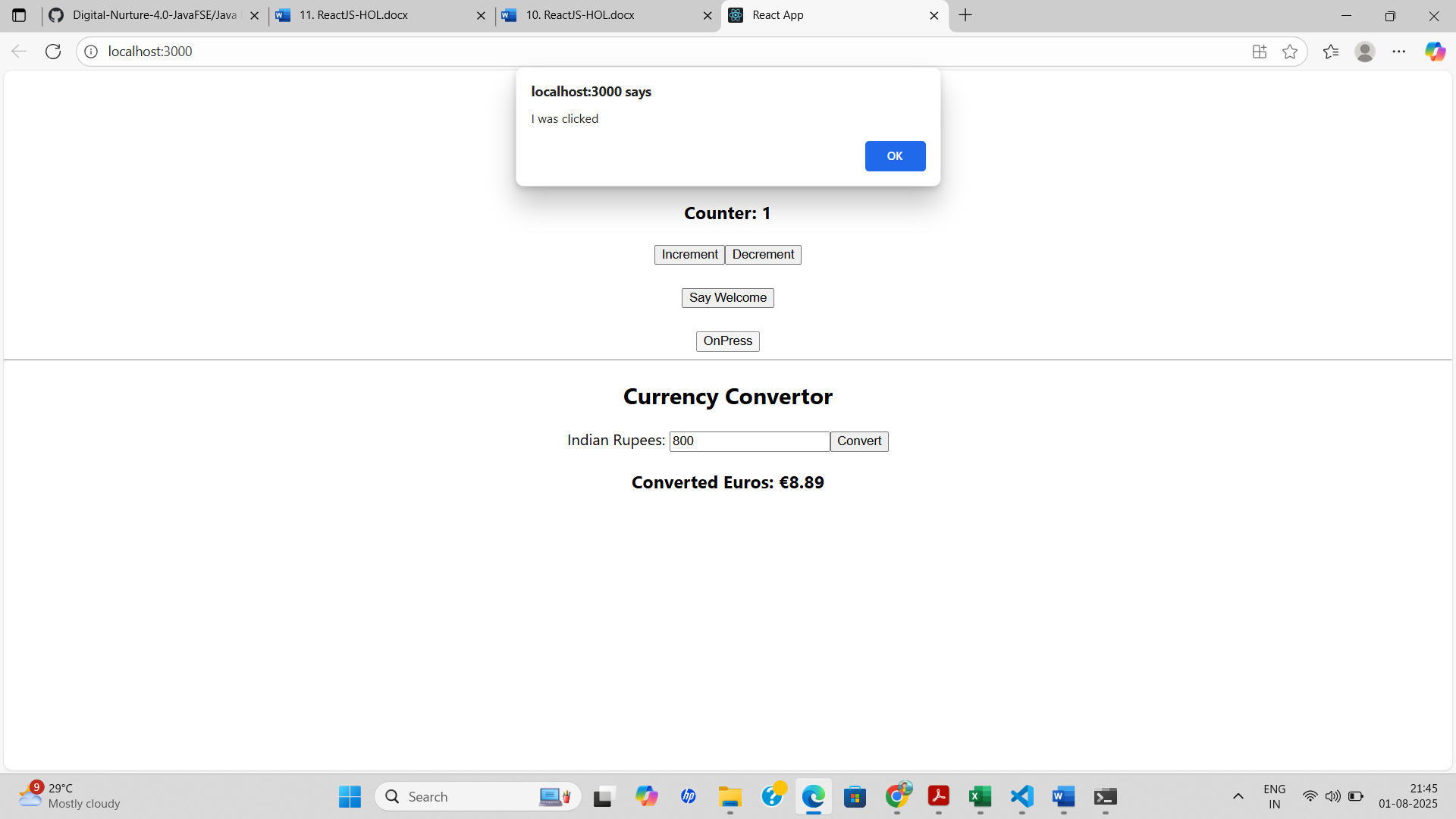
When ‘Decrement’ was clicked



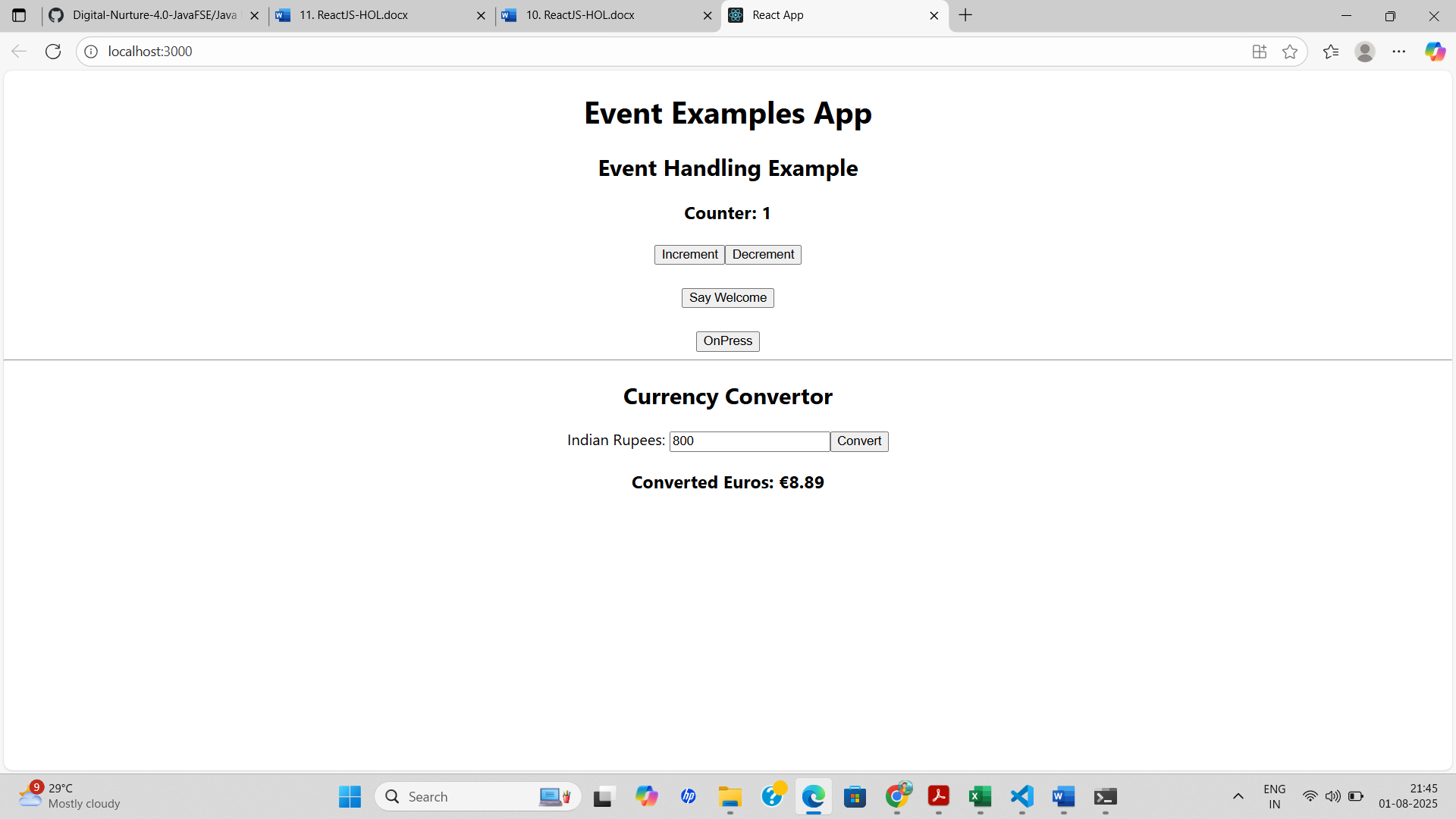
When ‘Say Welcome’ was clicked



When ‘OnPress’ was clicked



When ‘Convert’ was pressed



**12. REACTJS-HOL**

**Conditional Rendering in React:**Conditional rendering in React allows you to render different UI elements or components based on certain conditions. It works similarly to JavaScript conditions using if, else, ternary operators (condition ? true : false), or logical operators (&& or ||). This is useful to show or hide components or change content based on application state.

**Element Variables:**Element variables are used to store JSX elements. You can assign a JSX element to a variable and use it in the return() statement. This makes the code cleaner and easier to manage conditional logic.

Example

let message;

if (isLoggedIn) {

message = <h1>Welcome Back!</h1>;

} else {

message = <h1>Please Login.</h1>;

}

return <div>{message}</div>;

**Preventing Components from Rendering:**  
To prevent a component from rendering, you can return null instead of JSX. When a component returns null, nothing is rendered on the screen, but the component still exists in the component tree.

Example

function WarningBanner(props) {

if (!props.warn) {

return null;

}

return <div className="warning">Warning!</div>;

}

**APPROACH**

1. Create a React app using npx create-react-app ticketbookingapp and open it in VS Code.
2. Create two components: GuestPage to show flight details and UserPage to allow ticket booking.
3. Use React useState to manage the login state (isLoggedIn) and toggle between pages.
4. Implement Login and Logout buttons to update the login state and conditionally render content.
5. Use conditional rendering ({isLoggedIn ? <UserPage /> : <GuestPage />}) to switch views based on login status.

**CODE**

**Guest.js**

import React from 'react';

import FlightList from './FlightList';

function Guest({ onLogin }) {

return (

<div>

<h2>Welcome, Guest!</h2>

<button onClick={onLogin}>Login</button>

<FlightList />

</div>

);

}

export default Guest;

**User.js**

import React from 'react';

import FlightList from './FlightList';

function User({ onLogout }) {

return (

<div>

<h2>Welcome, User!</h2>

<button onClick={onLogout}>Logout</button>

<FlightList />

<button style={{ backgroundColor: "green", color: "white" }}>Book Ticket</button>

</div>

);

}

export default User;

**FlightList.js**

import React from 'react';

const flights = [

{ id: 1, name: 'Air India', from: 'Delhi', to: 'Mumbai' },

{ id: 2, name: 'IndiGo', from: 'Chennai', to: 'Bangalore' },

{ id: 3, name: 'SpiceJet', from: 'Hyderabad', to: 'Kolkata' },

];

function FlightList() {

return (

<div>

<h3>Flight Details</h3>

<ul>

{flights.map(flight => (

<li key={flight.id}>

✈ {flight.name} — {flight.from} to {flight.to}

</li>

))}

</ul>

</div>

);

}

export default FlightList;

**App.js**

import React, { useState } from 'react';

import Guest from './components/Guest';

import User from './components/User';

function App() {

const [isLoggedIn, setIsLoggedIn] = useState(false);

const handleLogin = () => setIsLoggedIn(true);

const handleLogout = () => setIsLoggedIn(false);

return (

<div style={{ textAlign: 'center', padding: '20px' }}>

<h1>✈️ Ticket Booking App</h1>

{isLoggedIn ? (

<User onLogout={handleLogout} />

) : (

<Guest onLogin={handleLogin} />

)}

</div>

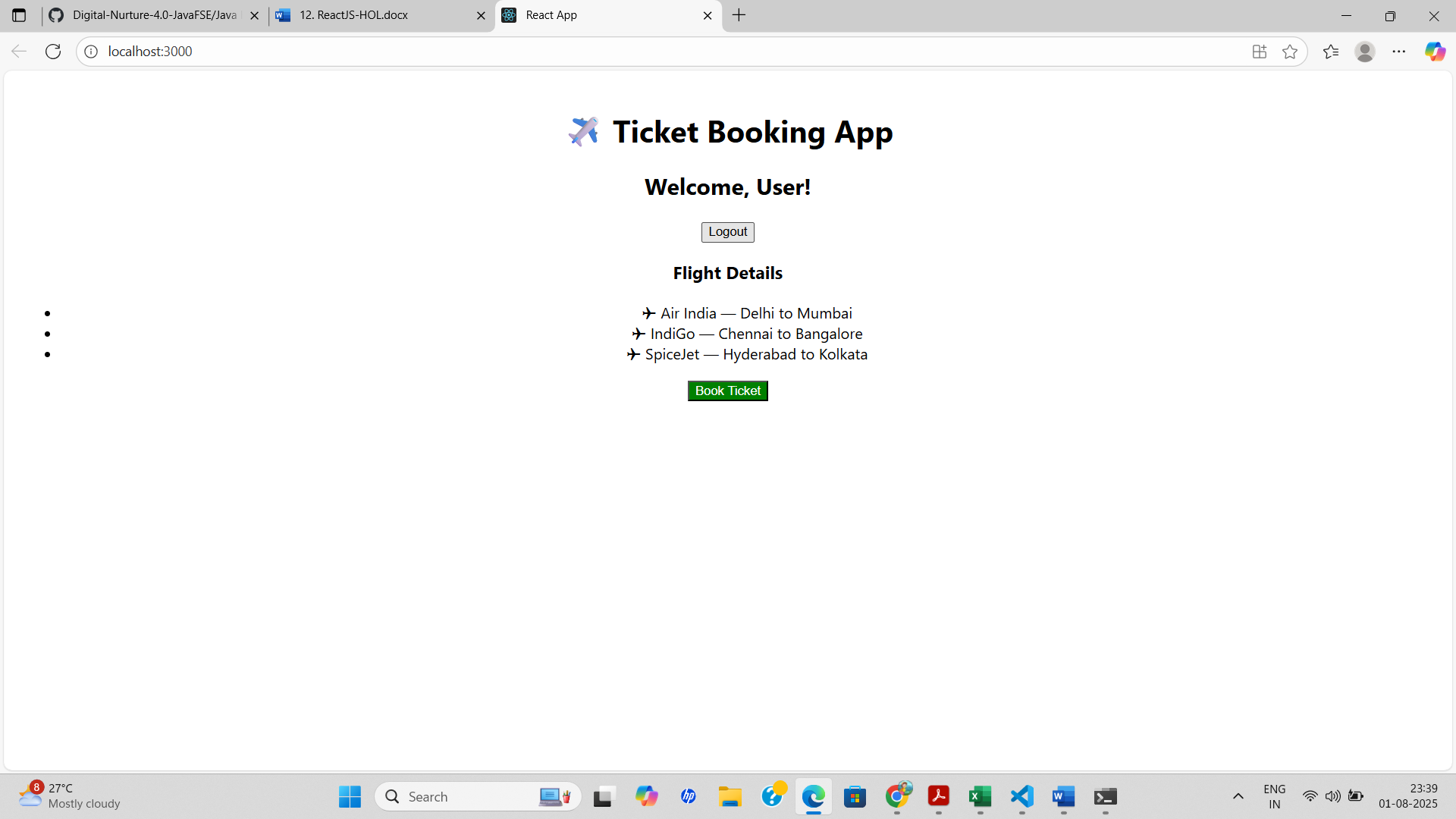
);

}

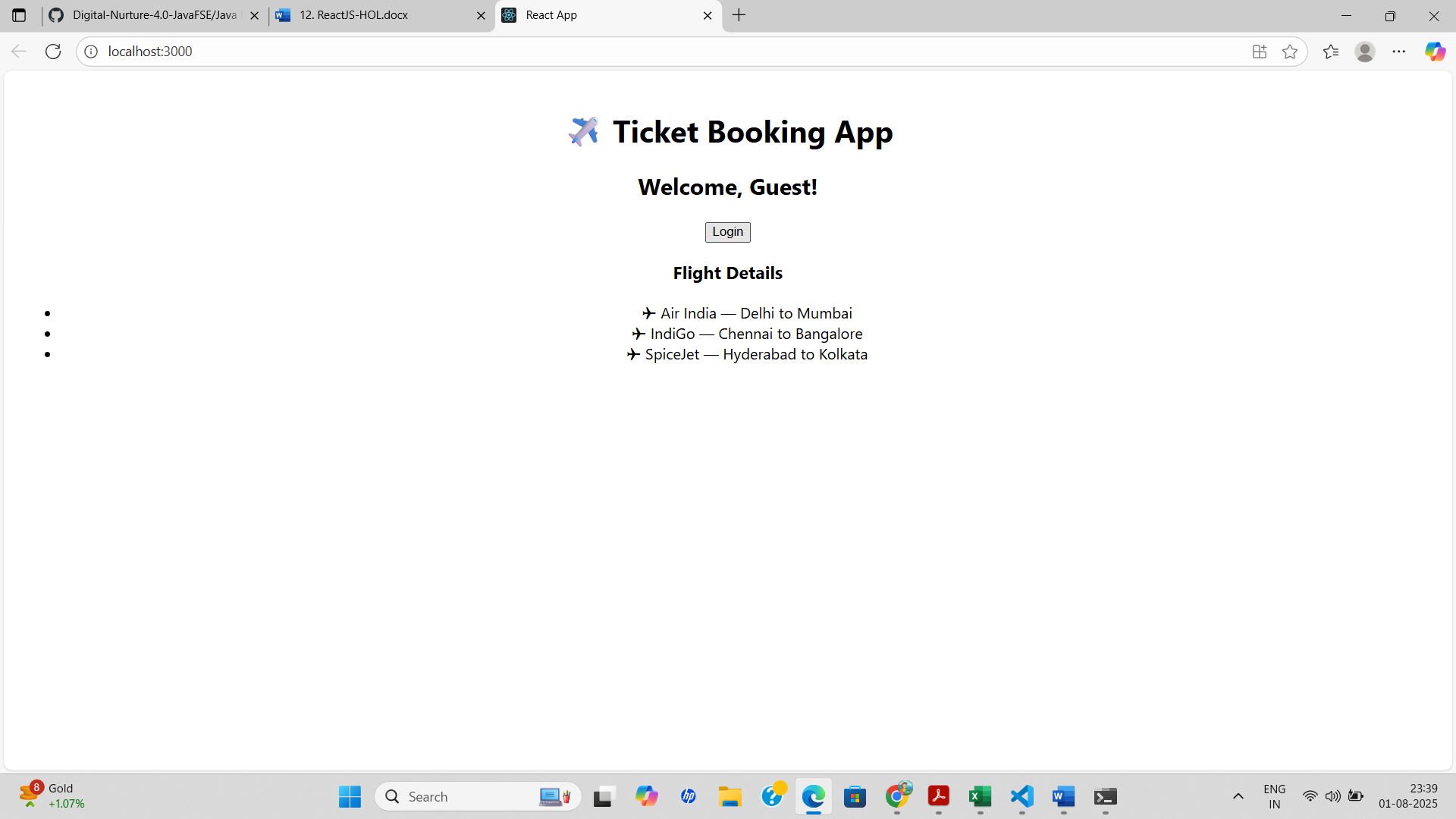
export default App;

**OUTPUT**

Login button → switches to user page with "Book Ticket"



Logout button → returns to guest view



**13. REACTJS-HOL**

**Explain various ways of conditional rendering**React provides several techniques to perform conditional rendering:

1. If-Else Statements – Use standard JavaScript if-else conditions inside the component function to return different JSX.
2. Ternary Operator – A concise way to render one of two elements: {condition ? <ComponentA /> : <ComponentB />}
3. Logical AND (&&) Operator – Used when you want to render something only if a condition is true: {isLoggedIn && <Dashboard />}
4. Element Variables – Assign components to a variable and render based on conditions before the return statement.
5. Immediately Invoked Function Expressions (IIFE) – Wrap conditional logic inside a function and call it inside JSX.

**Explain how to render multiple components**

You can render multiple components in React by placing them inside a parent component or returning them inside a <div>, <React.Fragment> or <>...</> fragment:-Example

return (

<>

<Header />

<Content />

<Footer />

</>

);

**Define list component**A list component is a reusable React component that displays a list of items. It uses the map() function to iterate through an array and render individual elements dynamically.

Example:

function NumberList(props) {

return (

<ul>

{props.numbers.map((num) => (

<li key={num.toString()}>{num}</li>

))}

</ul>

);

}

**Explain about keys in React applications**  
Keys help React identify which items have changed, are added, or are removed. They should be unique among siblings. Using keys improves performance and helps React render efficiently when dealing with dynamic lists.

**Explain how to extract components with keys**  
You can extract list items into their own components and pass a unique key prop when rendering.  
Example:

function ListItem(props) {

return <li>{props.value}</li>;

}

function NumberList(props) {

return (

<ul>

{props.numbers.map((num) => (

<ListItem key={num.toString()} value={num} />

))}

</ul>

);

}

**Explain React Map, map() function**  
In React, the map() function is used to iterate over arrays and return a list of elements. It helps dynamically render lists based on data.

Example:

const names = ['Alice', 'Bob', 'Charlie'];

const nameList = names.map((name) => <li key={name}>{name}</li>);

This approach is commonly used in rendering UI from an array of objects or strings.

**APPROACH**

1. Create a new React app using npx create-react-app bloggerapp and set up a basic folder structure with three components: BookDetails, BlogDetails, and CourseDetails.
2. In each component file, define and export a functional component displaying simple content like title and author/instructor.
3. In App.js, use the useState hook to manage the selected view (book, blog, or course) based on button clicks.
4. Implement all major conditional rendering techniques: if-else, ternary operator, logical &&, and element variables to render components based on the selected state.
5. Add buttons in App.js to switch between components dynamically, and display output using different rendering methods within the same app.

**CODE**

**BookDetails.js**

import React from 'react';

function BookDetails() {

return (

<div>

<h2>Book Details</h2>

<p>Title: React for Beginners</p>

<p>Author: John Doe</p>

</div>

);

}

export default BookDetails;

**BlogDetails.js**

import React from 'react';

function BlogDetails() {

return (

<div>

<h2>Blog Details</h2>

<p>Title: Exploring React</p>

<p>Author: Jane Smith</p>

</div>

);

}

export default BlogDetails;

**CourseDetails.js**

import React from 'react';

function CourseDetails() {

return (

<div>

<h2>Course Details</h2>

<p>Course: Full Stack Development</p>

<p>Instructor: Alex Johnson</p>

</div>

);

}

export default CourseDetails;

**App.js**

import React, { useState } from 'react';

import BookDetails from './components/BookDetails';

import BlogDetails from './components/BlogDetails';

import CourseDetails from './components/CourseDetails';

function App() {

const [selected, setSelected] = useState('book');

const renderComponentUsingIf = () => {

if (selected === 'book') return <BookDetails />;

if (selected === 'blog') return <BlogDetails />;

if (selected === 'course') return <CourseDetails />;

};

const elementVariable = (() => {

let component;

switch (selected) {

case 'book':

component = <BookDetails />;

break;

case 'blog':

component = <BlogDetails />;

break;

case 'course':

component = <CourseDetails />;

break;

default:

component = null;

}

return component;

})();

return (

<div style={{ textAlign: 'center', marginTop: '50px' }}>

<h1>Blogger App</h1>

<div>

<button onClick={() => setSelected('book')}>Show Book</button>

<button onClick={() => setSelected('blog')}>Show Blog</button>

<button onClick={() => setSelected('course')}>Show Course</button>

</div>

<hr />

<h3>Conditional Rendering - if/else</h3>

{renderComponentUsingIf()}

<h3>Conditional Rendering - Ternary Operator</h3>

{selected === 'book' ? <BookDetails /> : selected === 'blog' ? <BlogDetails /> : <CourseDetails />}

<h3>Conditional Rendering - Element Variable</h3>

{elementVariable}

<h3>Conditional Rendering - && Operator</h3>

{selected === 'book' && <BookDetails />}

{selected === 'blog' && <BlogDetails />}

{selected === 'course' && <CourseDetails />}

</div>

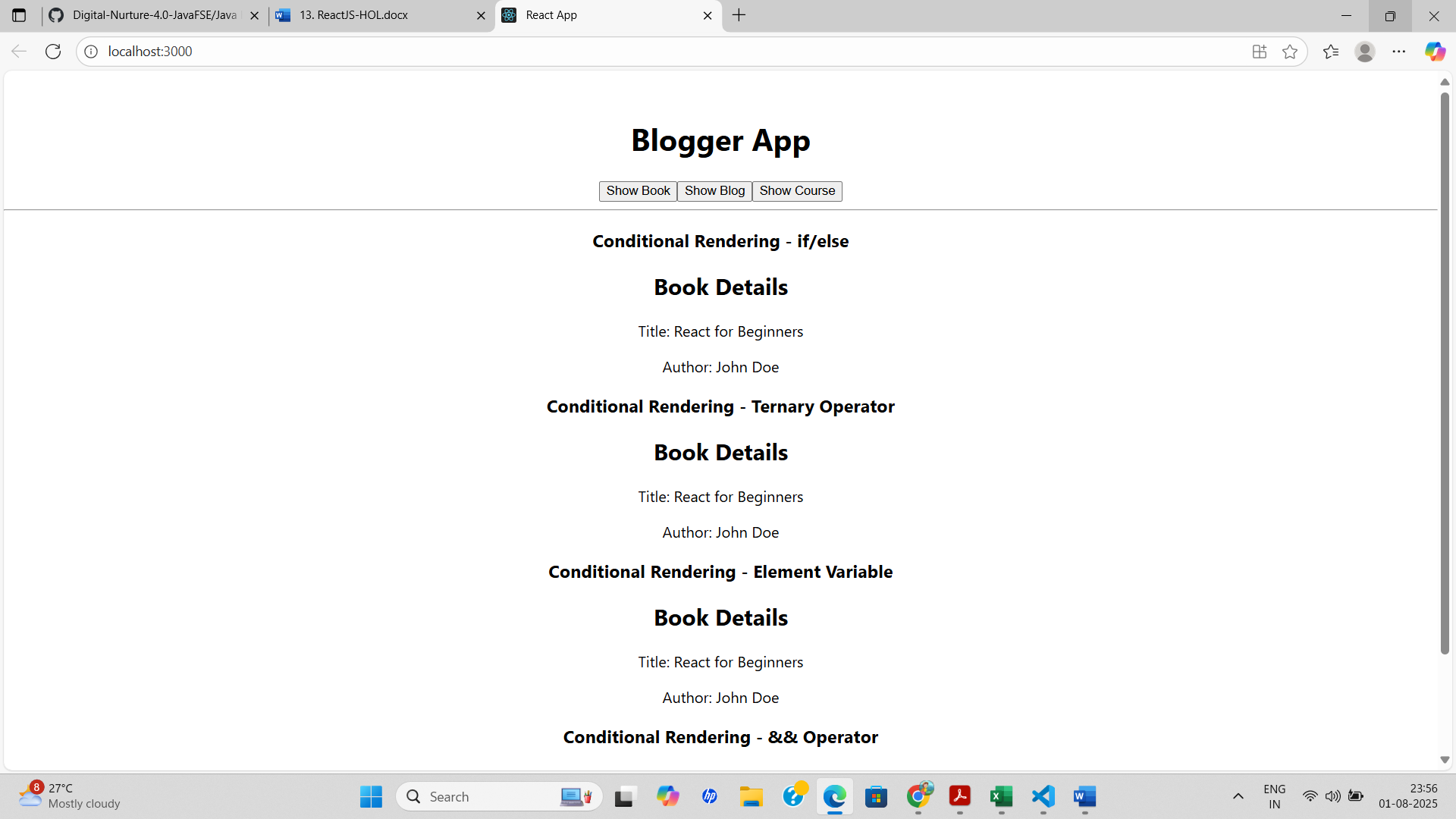
);

}

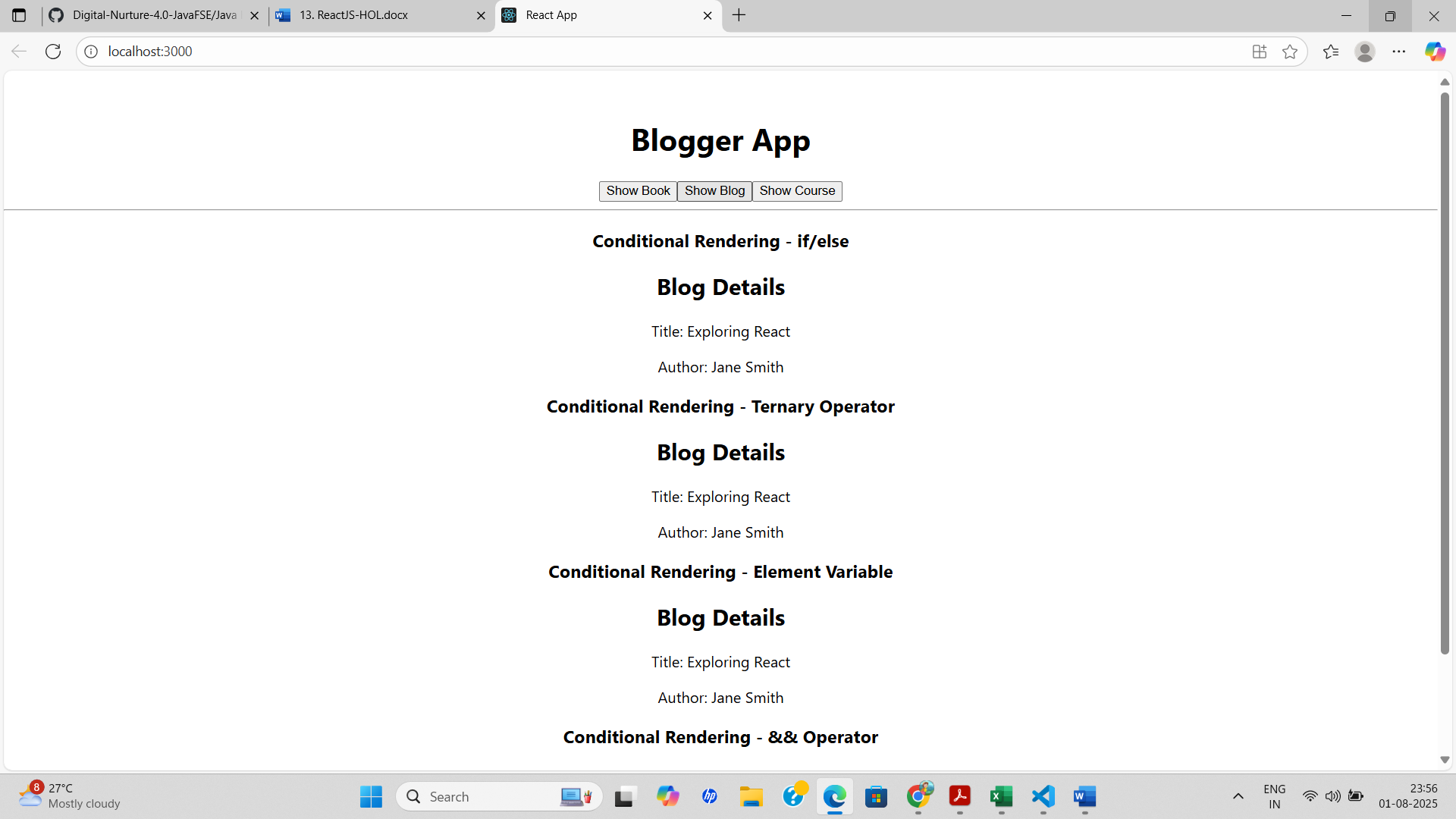
export default App**;**

**OUTPUT**

When ‘Show Book’ is clicked



When ‘Show Blog’ is clicked



When ‘Show Course’ is clicked

